Jessica Ramirez

CS330

February 22, 2025

Professor Gray

Design Decisions

In designing this 3D scene, I aimed to create a visually appealing yet computationally efficient environment. I chose a small set of everyday objects—a sphere (ball), a cube (box), a mug, and a spoon—because they represent both basic geometric forms and commonly recognizable items. By including these different shapes, I can showcase multiple mesh types (e.g., sphere, cylinder, torus) and demonstrate how each shape interacts with lighting and textures. For instance, the sphere highlights spherical mapping, the box illustrates multiple textured sides, and the mug with a handle (using a half torus) demonstrates more complex geometry. A plane mesh serves as the “table” or ground surface, giving a stable reference point for the scene and allowing me to experiment with UV scaling on a tiled texture. I also added a backdrop plane behind the objects to frame the composition and show how simple geometry can be used to create a clean background. Each object is associated with a specific material (e.g., plastic, metal, tile) that I defined in the **DefineObjectMaterials()** function, which allows for varying specular highlights, shininess, and diffuse color, showcasing how material properties influence the final render.

From a performance standpoint, each mesh (plane, sphere, box, cylinder, torus) is loaded only once in memory, but can be reused multiple times with different transformations and textures, ensuring efficient memory usage and reducing overall rendering overhead. The **LoadSceneTextures()** function similarly loads texture images once, binding them to texture slots for easy reference in subsequent drawing operations. Although the provided code primarily focuses on scene management and rendering, I typically pair it with an input-handling system for first-person or third-person navigation, allowing users to move around with **W**, **A**, **S**, **D** keys or arrow keys, plus **Q** and **E** for vertical movement, while mouse movement adjusts the camera’s yaw and pitch.

To keep the code organized and maintainable, I used a modular design within the **SceneManager** class. For instance, **CreateGLTexture()** encapsulates the logic for loading image files (via **stb\_image**), generating OpenGL texture objects, and setting parameters such as wrap mode and filtering, making it reusable for any new image-based textures. The **SetTransformations()** function combines scaling, rotation, and translation into a single model matrix, allowing me to apply transformations to any mesh without duplicating code. **SetShaderTexture()** and **SetShaderMaterial()** similarly abstract away texture-slot binding and uniform variable setup, so swapping textures or materials requires only one function call. Finally, **RenderScene()** demonstrates how these functions integrate: by calling **SetTransformations()**, **SetShaderTexture()**, and **SetShaderMaterial()** in sequence, each object is positioned, textured, and shaded appropriately, making the code more readable, easier to debug, and simpler to extend.

Overall, this modular design ensures efficient memory usage, supports straightforward navigation controls, and simplifies future expansion of the 3D scene